Formal Development for Railway Signaling Using Commercial Tools

Formal Development for Railway Signaling Using Commercial Tools

Alessio Ferrari[1], Alessandro Fantechi[2]
Stefano Bacherini[1], and Niccoló Zingoni[1]

[1] General Electric Transportation Systems (GETS), Firenze, Italy
[2] Universitá di Firenze, DSI, Firenze, Italy

Abstract. This report presents the approach experimented by a railway signaling manufacturer for the development of applications through Simulink/Stateflow in a standard–regulated industrial framework.

The General Electric Transportation Systems (GETS) railway signaling division of Florence, inside a long-term effort of introducing formal methods to enforce product safety, decided to adopt the Simulink/Stateflow tool-suite to exploit model based development and code generation within its own development process [1]. Products traditionally provided by GETS, like any railway signaling application developed for Europe, shall comply with the CENELEC norms [2]. Introducing the Simulink/Stateflow tool-suite within a CENELEC based process is not a straightforward step, and GETS faced two crucial obstacles: the lack of a formal semantics for the Simulink/Stateflow languages, and the absence of a CENELEC compliant code generator.

The languages used by Simulink and Stateflow are not formally specified and their semantics is essentially given by the simulation engine itself. This increases the difficulty of defining an effective formal verification strategy, a highly recommended practice according to the CENELEC norms.

Code generators provided for the tool-suite (in particular Stateflow Coder) are not certified for railway software development, this complicating their adoption in this domain. In order to defeat these problems, GETS first introduced a set of modeling guidelines to restrain the semantics of the tools [3]. The idea is based on the intuition that reducing the Simulink/Stateflow languages to a semantically unambiguous subset enables proper code synthesis and formal verification. Once developed this set of modeling rules, a proper strategy including formal development, model based unit testing and formal verification of modules has been defined. Given a set of system-level functional requirements, these can be partitioned into separate sets of unit requirements and then formalized into Stateflow models according to the GETS guidelines. Each model represents an independently verifiable system component. Unit testing based on requirement coverage is then performed on the models through the Simulink environment, and during test execution a test observer is used to register the test-suite input data and the test results. The registered test-suite is executed on the auto-coded unit and results are automatically compared. Finally, the module is analyzed through the

Polyspace tool for abstract interpretation, in order to increase the confidence on the correctness (i.e., runtime errors freedom) of the generated code (Fig. 1). This strategy basically settles the problem of having a qualified code generator, since certification of conformity is ensured each time code is synthesized from a model. Verification of functional requirements is provided at Stateflow chart level: unit requirements are translated into formulas made of Simulink blocks and validated against the Stateflow model through the property proving engine called Simulink Design Verifier (fig. 2).

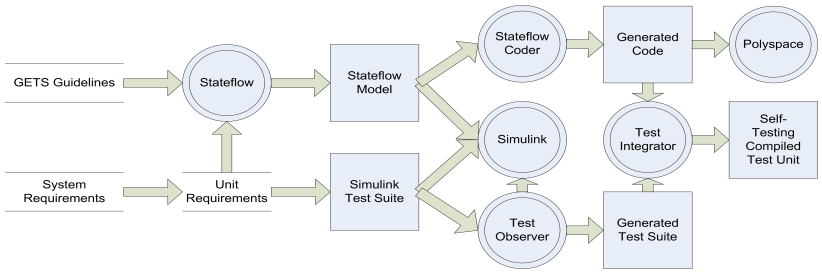

Fig. 1. Overview of our strategy for model based testing

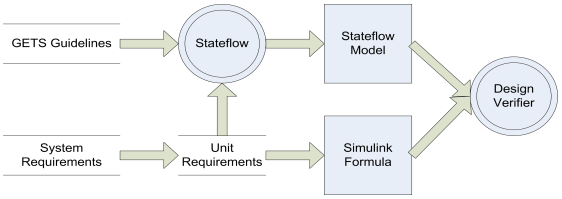

Fig. 2. Overview of our strategy for formal verification

The presented approach is focused on the level of system modules, since the strategy has been fully put into practice only at this level during the development of the logic of an Automatic Train Protection system called SSC/SCMT BaseLine 3 (150K LOC of auto-generated code). Extension of the approach at the overall system level is theoretically feasible, but we are still working on strategies for putting it into practice in the most effective manner.

References

1. Bacherini, S., Fantechi, A., Tempestini, M., Zingoni, N.: A Story about Formal Methods Adoption by a Railway Signaling Manufacturer. FM 2006. LNCS, 4025/2006, Hamilton, Canada, (2006).
2. European Committee for Electrotechnical Standardization: CENELEC EN 50128, Railway Applications - Software for Railway Control and Protection Systems. (1997).
3. Ferrari, A., Fantechi, A., Bacherini, S., Zingoni, N.: Modeling Guidelines for Code Generation in the Railway Signaling Context. Proceedings of 1st NASA Formal Methods Symphosium (NFM). Moffet Field, California, U.S.A., (2009).